Published in Better Programming · Follow

bytefish · Follow
Jan 27 · 8 min read ★

10 CSS Tricks That Greatly Improve User Experience

Spend 1 hour to learn, and enhance your web app forever.

A successful Web App must have a good user experience. What do you think of when we talk about improving the user experience?

In fact, there is a point that is easy to be ignored by developers, that is CSS. We can use some CSS tricks to improve the presentation, interaction details, and accessibility of a webpage.

And these tricks don't cost much and don't consume server resources. You only need to spend two hours learning, then you can apply it to all your projects, and improve user experience forever.

Clickable area

Sometimes your button is small, which may cause users to not be able to click the button accurately. This phenomenon often occurs on mobile phones.

It can be very frustrating for users if they click too many times and don't click the button they want, or click the wrong button.

So how to solve this problem? Some developers might say: make the button bigger.

But the sizes of elements in webpages are often fixed, we cannot easily adjust the size of an element. And if the button is too big, it feels weird.

A better solution is to increase the clickable area of the button without changing its original size. Specifically: we can use pseudo-elements to increase the clickable area of an element.

For example, here is a button:

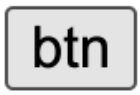
```
<button id="btn">btn</button>
```

Then we can add a pseudo-class for it:

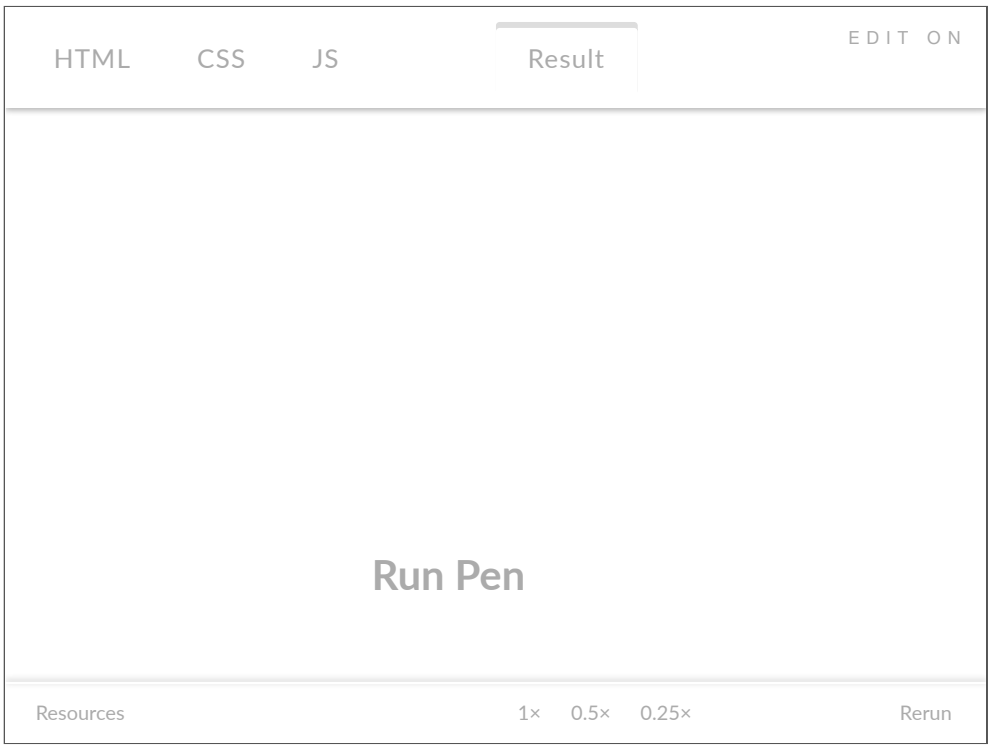


```
    position: absolute;
    top: -20px;
    right: -20px;
    bottom: -20px;
    left: -20px;
  }
```

At this time, if we click the area around the button, we can still trigger the click event of the button.

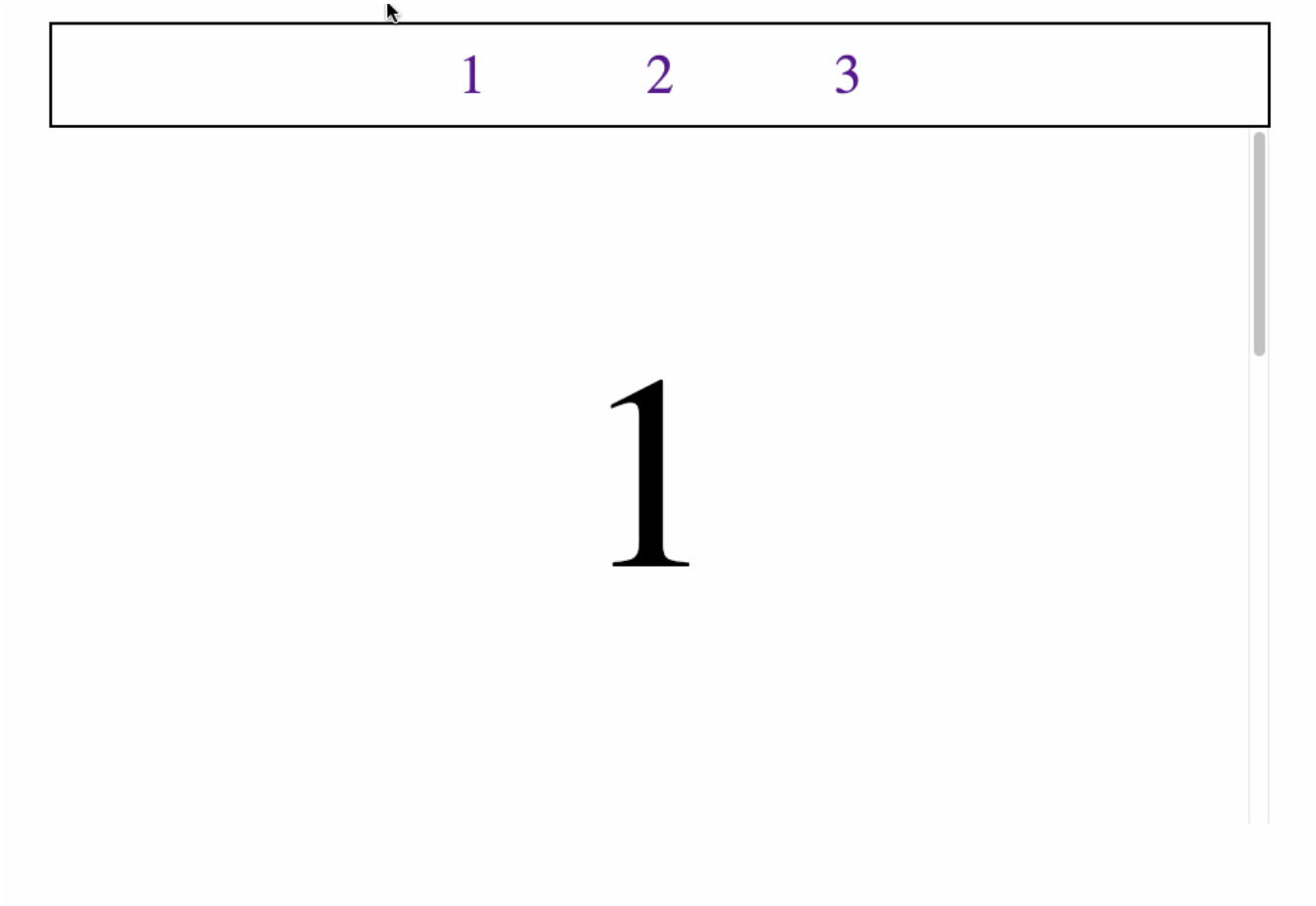


Codepen demo:

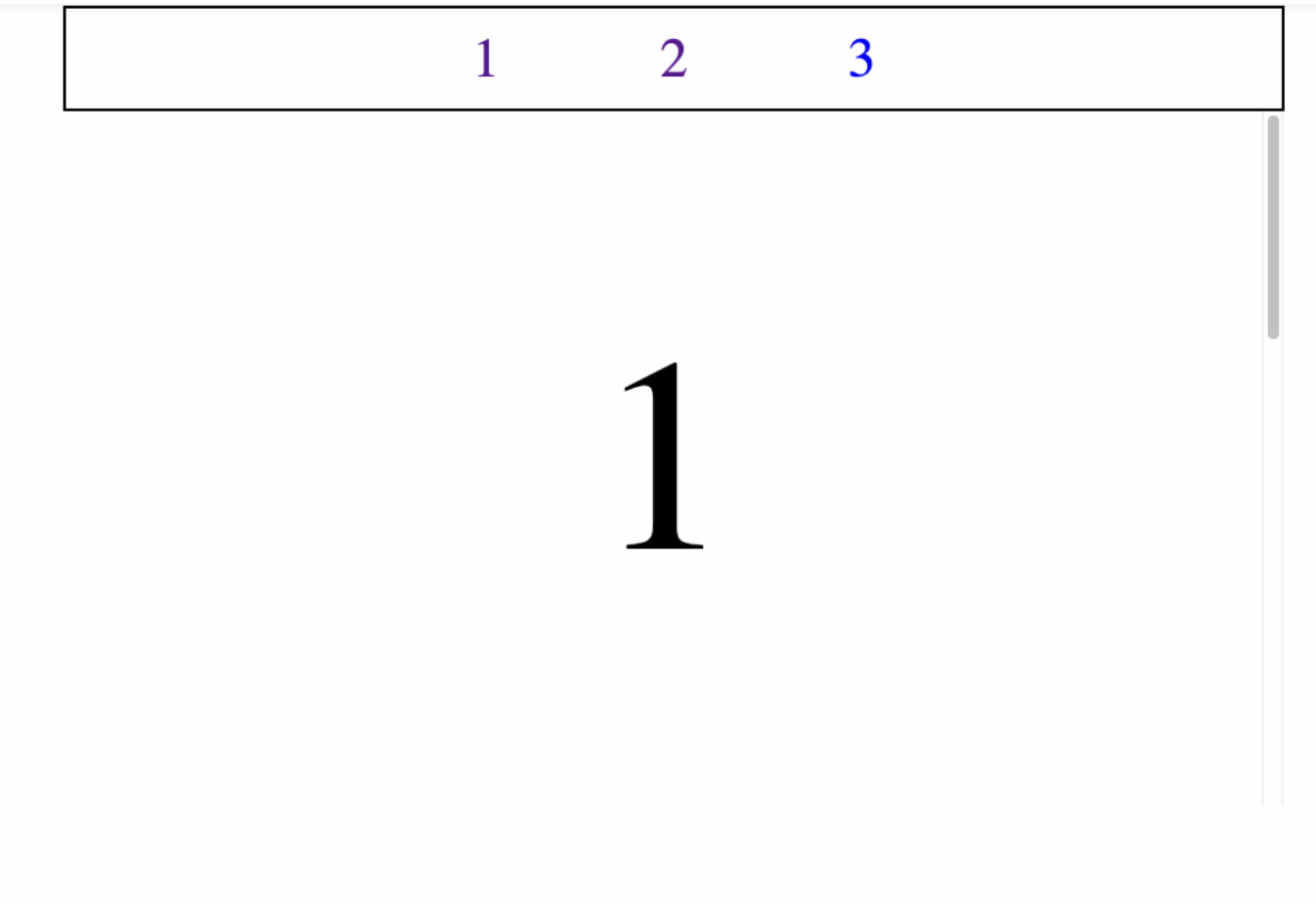


Smooth Scroll

When the page is scrolled by `#` link, the default effect is like this:



This sudden jump can make people feel uncomfortable. To solve this problem, we can use this CSS style: `sroll-behavior: smooth`.

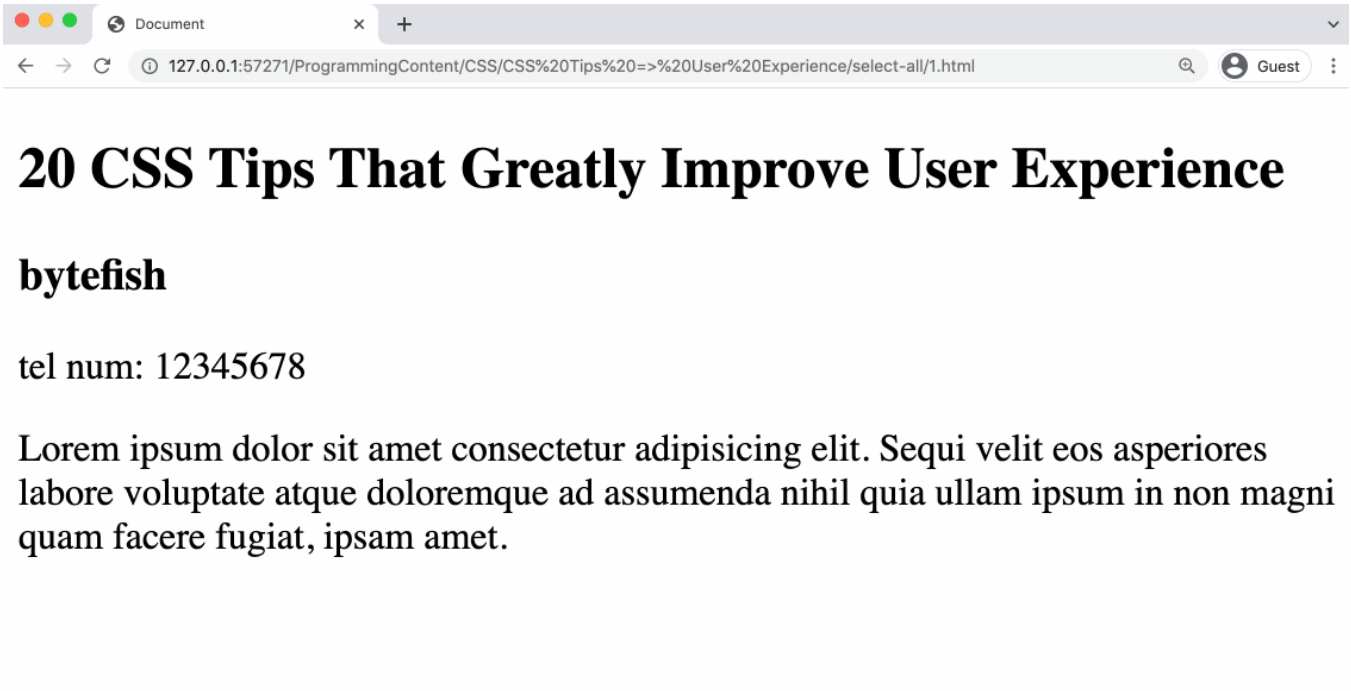


CodePen demo:

Select all text

Our pages often need to provide some content for users to select, such as phone numbers, addresses, titles, etc. And these texts should be a whole, we hope that when the user clicks on part of the text, the remaining text will be automatically selected.

For example:



To achieve this effect is very simple, just use this CSS style: `user-select: all` . The **`user-select`** CSS property controls whether the user can select text. If its value is `all` , means all the content of an element shall be selected together atomically.

CodePen demo:

If you want to add some extra styles after the text is selected, you can use `::selection` . The **`::selection`** CSS pseudo-element applies styles to the part of a document that has been highlighted by the user (such as clicking and dragging the mouse across text).

But you should remember: Only certain CSS properties can be used with `::selection` :

- `color`
- `background-color`
- `text-decoration` and its associated properties
- `text-shadow`
- `stroke-color` , `fill-color` and `stroke-width`

CodePen demo:



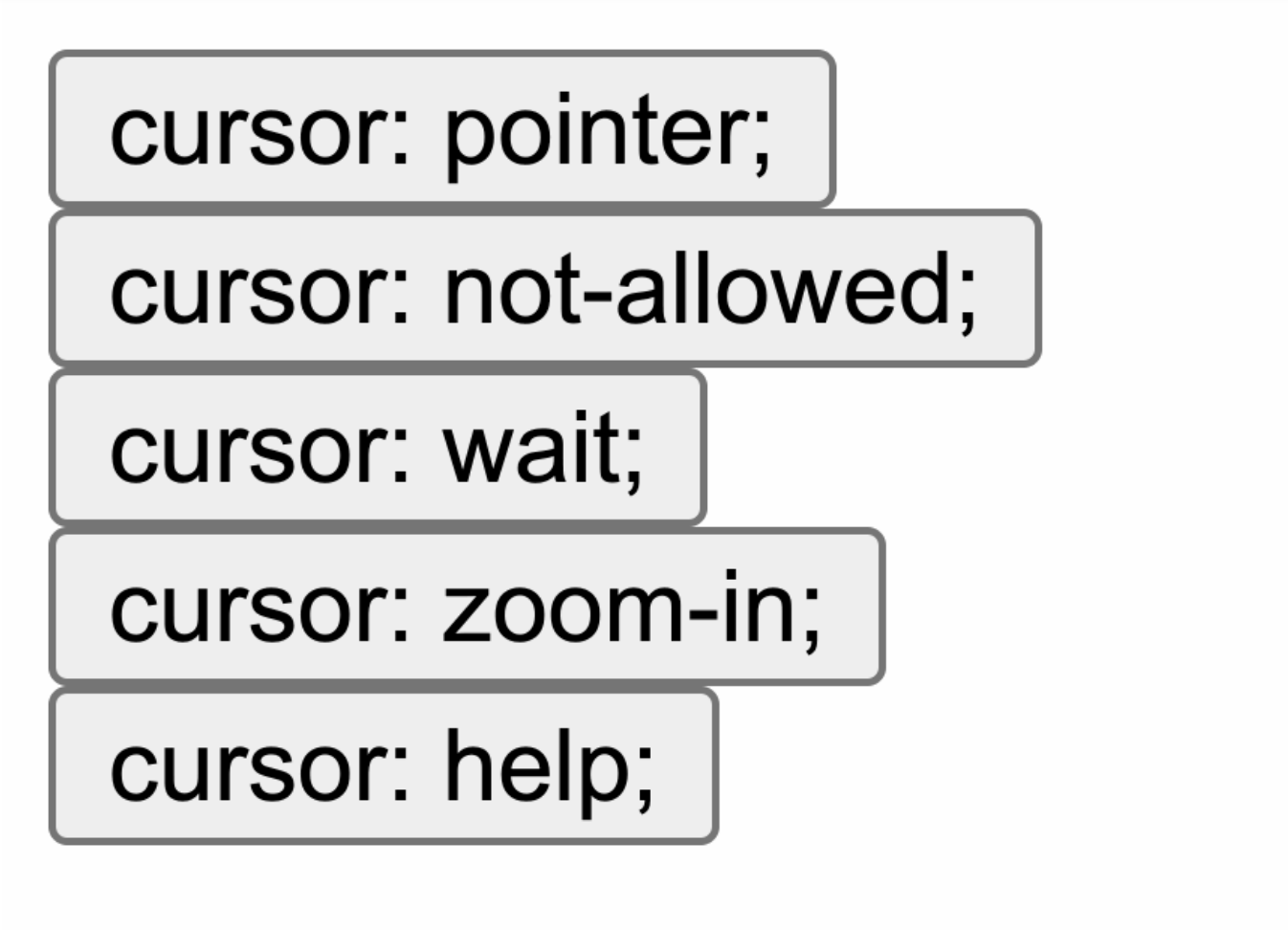
Cursor

Using different mouse styles in different scenarios can help readers to perceive the current state of the page, thereby improving the user's interactive experience.

The `cursor` CSS property sets the mouse cursor, if any, to show when the mouse pointer is over an element.

The cursor setting should inform users of the mouse operations that can be performed at the current location, including text selection, activating help or context menus, copying content, resizing tables, and so on. You can specify either the *type* of cursor using a keyword, or load a specific icon to use (with optional fallback images and mandatory keyword as a final fallback).

Example:



CodePen demo:

There are many cursor styles, you can find them all on the MDN documentation.

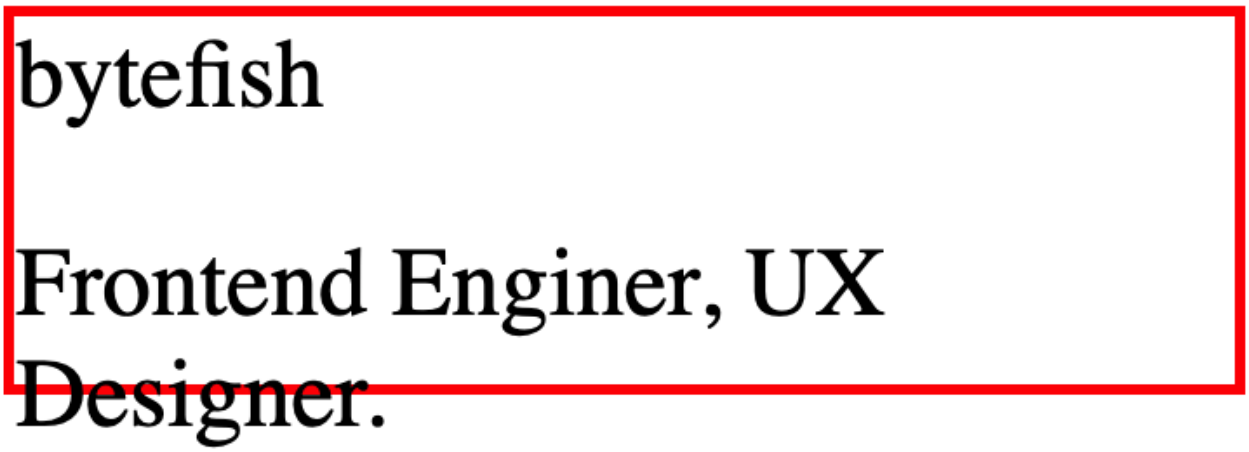
Text Overflow

Now let's look at the problem of text-overflow. If the content of a text container is returned from the server, or entered by the user, it is difficult to predict how long the text will be.

Without any precautions, you might write code like this:

This container has a fixed width and height and wraps the name and bio element.

But if some users' bios are too long, it will cause the text to overflow the container and make the page looks bad.



At this point, we can collapse the overflowing text. Doing this is as simple as adding three lines of CSS styles.

```
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
```

the end of the text to indicate there is some hidden text to users.

Codepen demo:

developer to keep the image at a fixed size. Then you write code like this:

And the webpage looks like this:

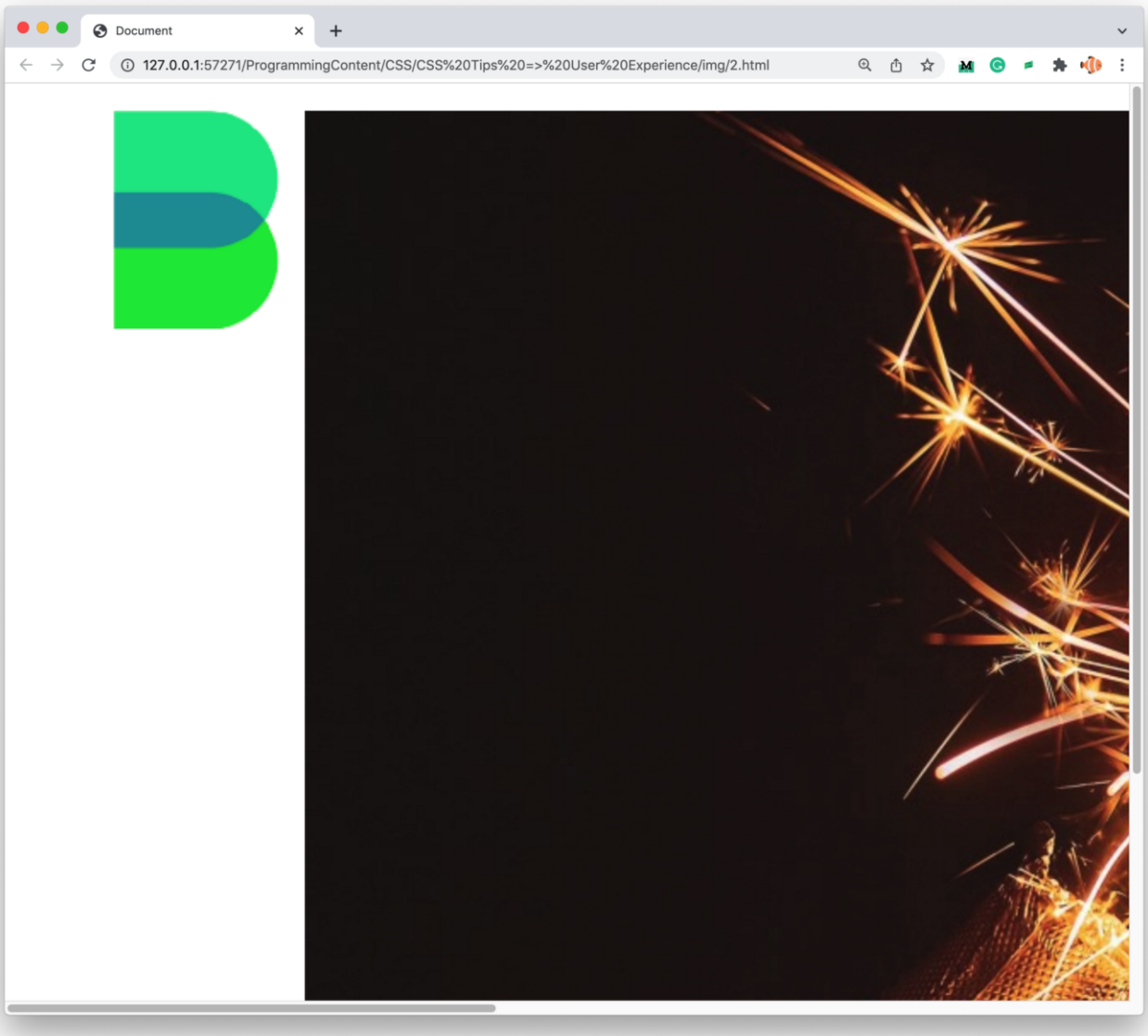
The pictures are arranged as you would expect them to look.

Normally there is no problem. But when we write code, we can't assume that everything will develop as we expect. We need to be fully prepared. If the image returned by the backend is abnormal and does not meet the expected size, it may be large or small, then your layout will be disrupted.

You can replace the link to one of the images with this:

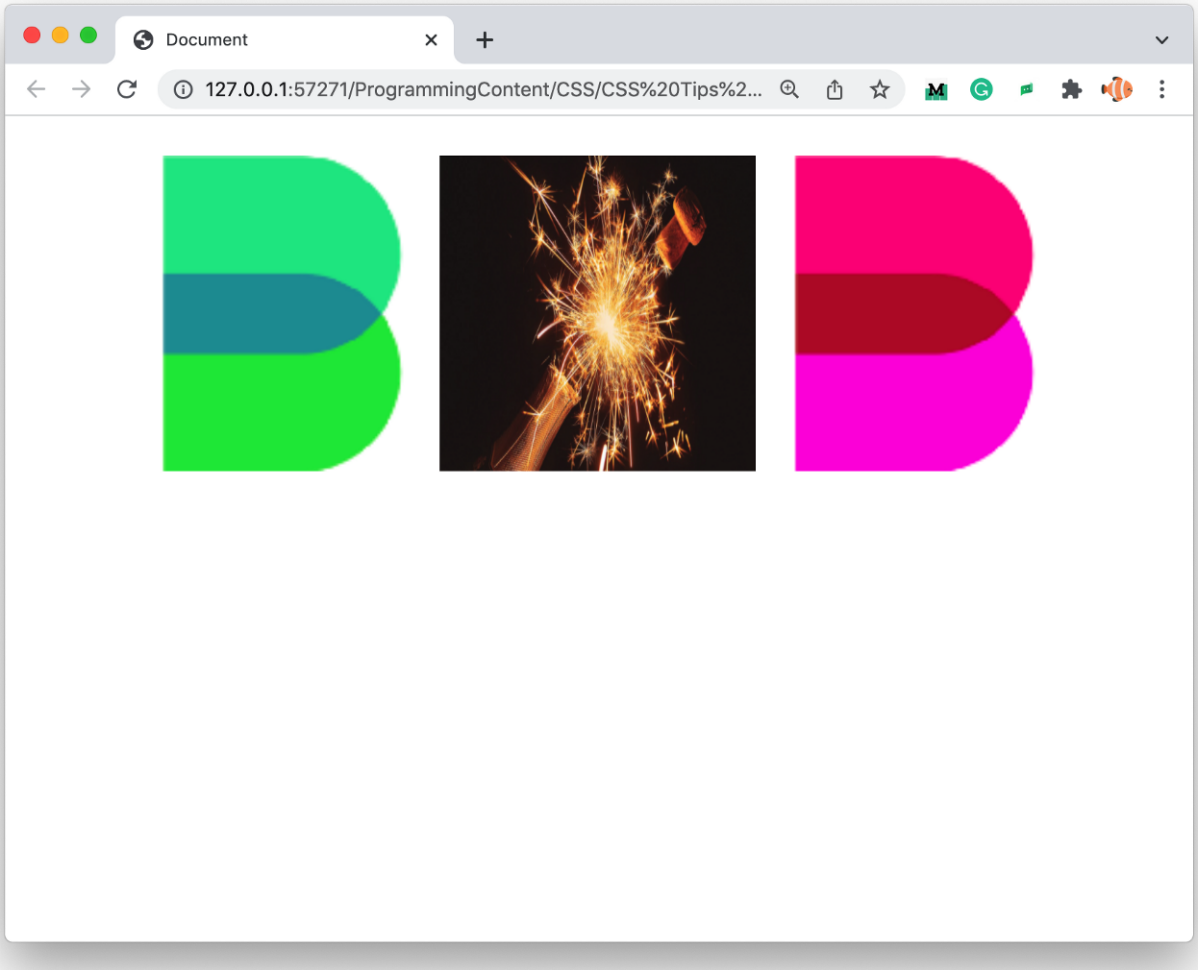
```
https://miro.medium.com/max/1400/0*zQaS0awtSTOO-JYa.jpg
```

You will find that the page suddenly becomes cluttered:



In order to prevent this problem and make our page more robust, we can set the width and height of the image. In this way, we don't have to worry about the size of the image returned by the backend.

```
img {
  width: 128px;
  height: 128px;
}
```

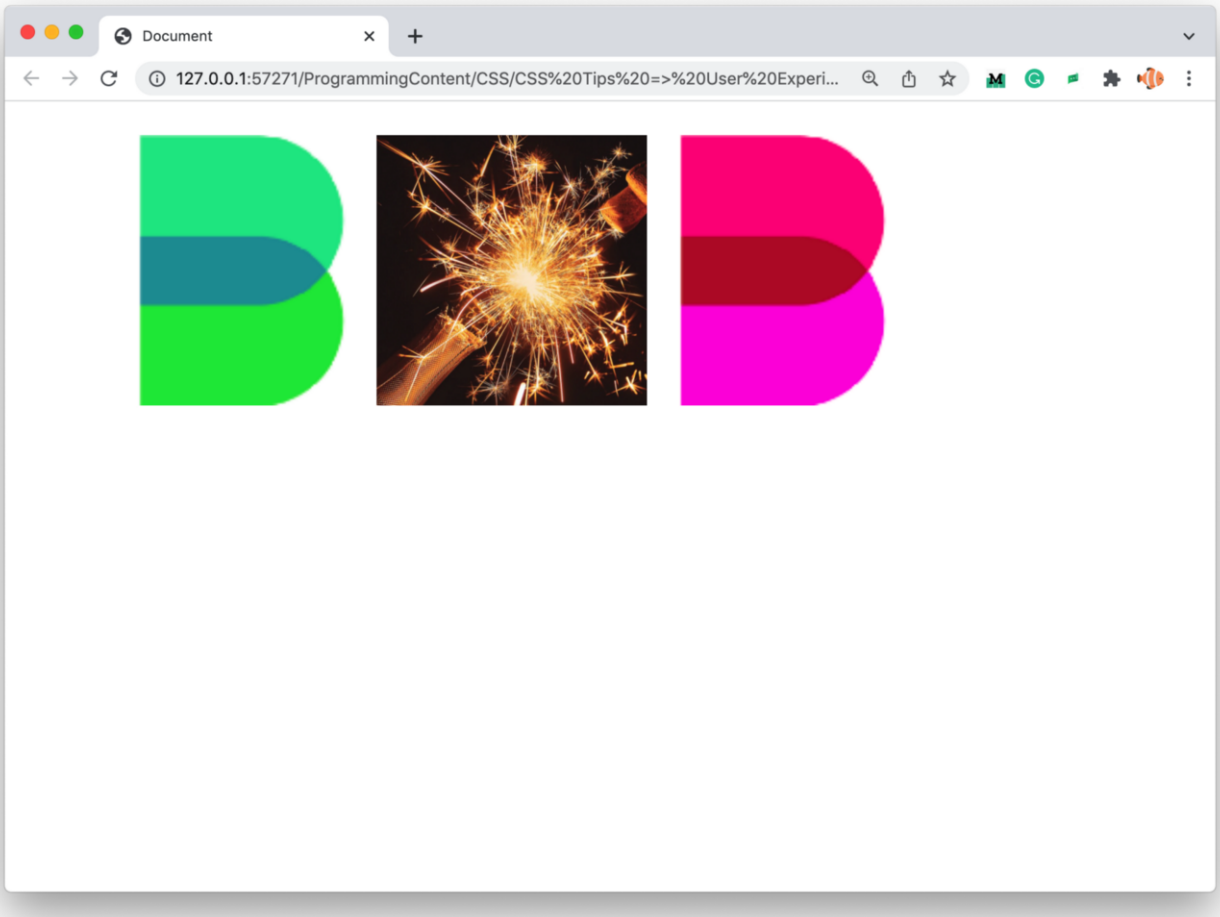


But the above writing has a disadvantage: if the aspect ratio of the image itself does not match the aspect ratio we set, the image will be compressed or stretched.

In order to keep the image in its original aspect ratio, we can use `object-fit: cover;` .

```
img {
  width: 128px;
  height: 128px;
  object-fit: cover;
}
```

The **`object-fit`** CSS property sets how the content of a replaced element, such as an `<img>` or `<video>` , should be resized to fit its container.



No image

The situations we discussed before are based on the premise that we can get the image. However, in real applications, it may be due to the instability of the back-end service, or the user's own network signal is not good, our web page may not be able to load images correctly.

When the image is missing, the browser's default style is not elegant, here we can optimize it.

We can add an `onerror` event to the `img` element. If there is an error while loading the image, then we can add a style to the element via the `onerror` event and use the 404 image.

`img` element:

Suppose this is our 404 image:

```
https://cdn-images-
1.medium.com/max/1600/1*we8wfyztsdo12e2Cww6oVA.jpeg
```

https://www.freepik.com/free-vector/404-error-with-person-looking-concept-illustration_20824298.htm#query=404&position=12&from_view=keyword

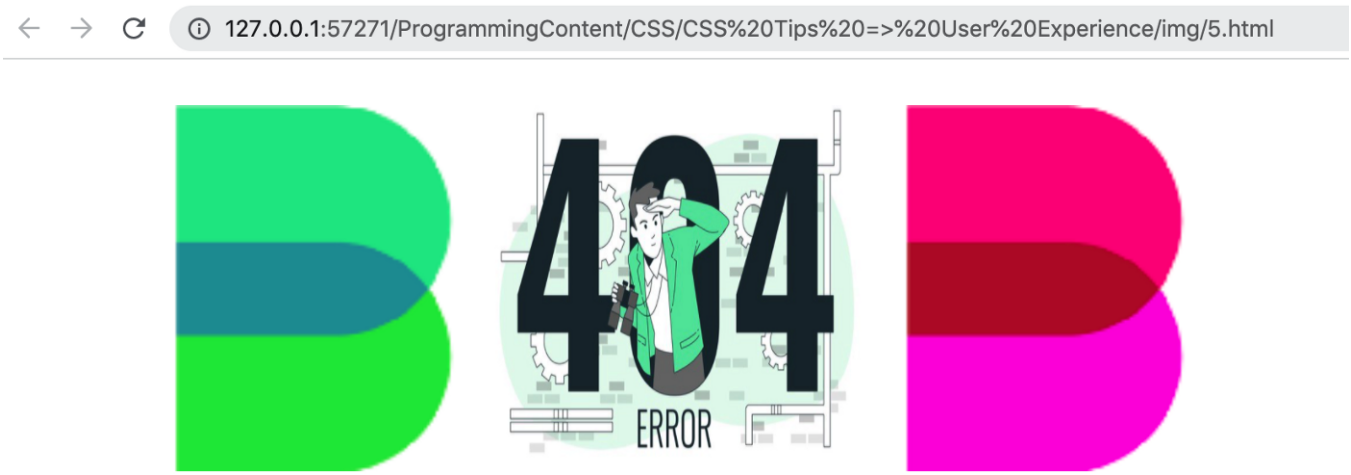
This is the CSS style:

```
img.error {
      display: inline-block;
      transform: scale(1);
      content: '';
      color: transparent;
    }

img.error::before {
      content: '';
      position: absolute;
```

```
1.medium.com/max/1600/1*we8wfyztsdo12e2Cww6oVA.jpeg') no-repeat
center / 100% 100%;
    }
```

In this way, when the image link in the `img` element cannot load the image, our 404 image will be used.



There is one more point to optimize here. In this case, if the original image is not loaded correctly, the user has no idea what the image should have been. In order to facilitate the user's understanding, we can display the `alt` attribute of the `img` element to the page.

Code:

```
img.error::after {
      content: attr(alt);
      position: absolute;
      left: 0;
      bottom: 0;
      width: 100%;
      line-height: 2;
      background-color: rgba(0, 0, 0, .5);
      color: white;
      font-size: 12px;
      text-align: center;
      white-space: nowrap;
      overflow: hidden;
```

Suppose the `img` has the `alt` attribute like this:

```
<img src="https://miro.medium.com/xxx.jpg" alt='Log of Medium' >
```

Then it will look like this:

CodePen demo:

Color Contrast

When you are designing color combinations, have you considered the color contrast of the page?

You need to know that there are many color blind and color weak users in the world. If the contrast of your page is low, it may cause them to be unable to use your product normally. Whether it's out of a humanistic concern, or out of keeping your clients, you should design for the right contrast.

The WCAG AA specification states that all the important content needs to have a color contrast ratio of `4.5:1` or more.

Here is a tool for Contrast Checker:

Contrast Checker

The five boxing wizards jump quickly. The five boxing wizards jump quickly. Enter a foreground and background color in...

webaim.org

Example:



Contrast Checker

Home > Resources > Contrast Checker

Foreground Color

#035E8C

Lightness

↔

Background Color

#FFF5F5

Lightness

Contrast Ratio

6.57:1

permalink

Normal Text

WCAG AA: **Pass**

WCAG AAA: **Fail**

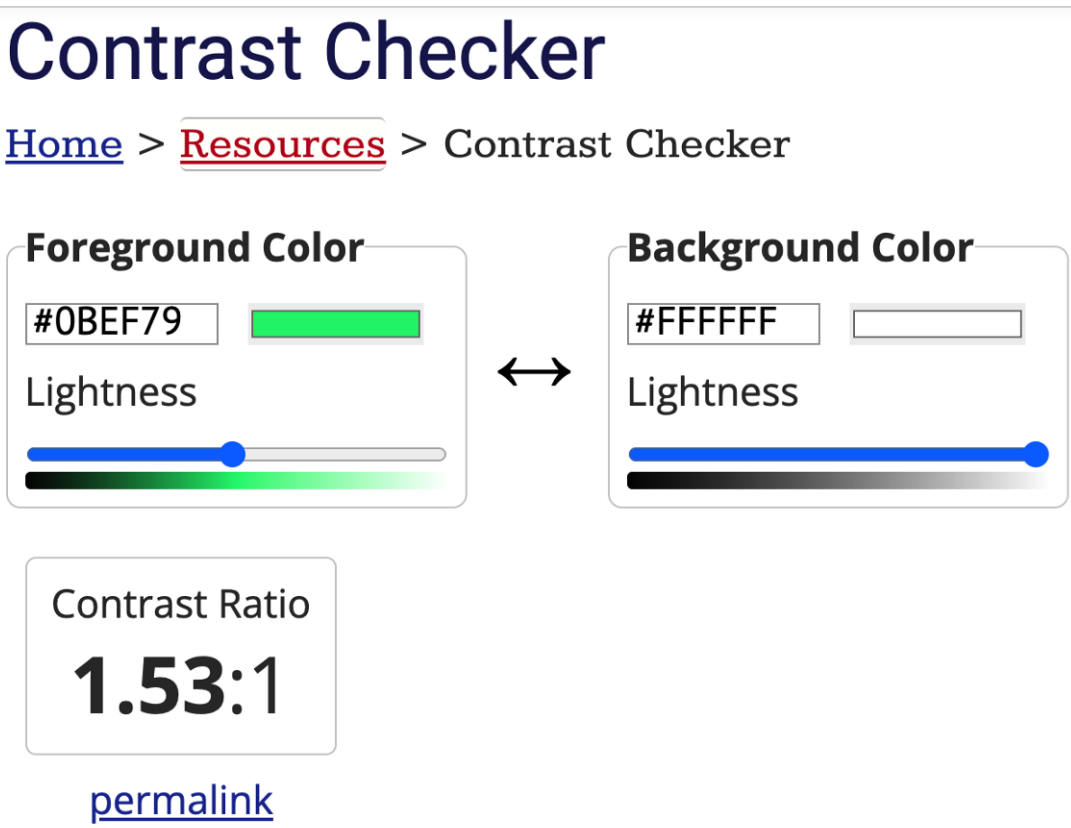
The five boxing wizards jump quickly.

Large Text

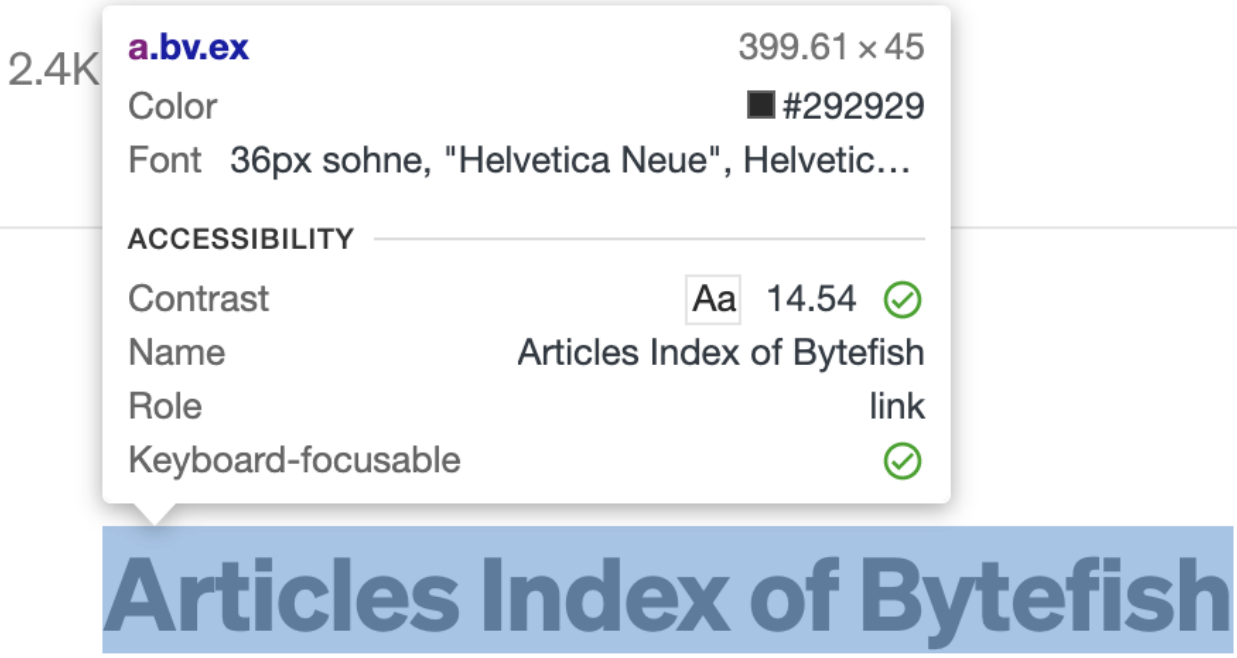
WCAG AA: **Pass**

WCAG AAA: **Pass**

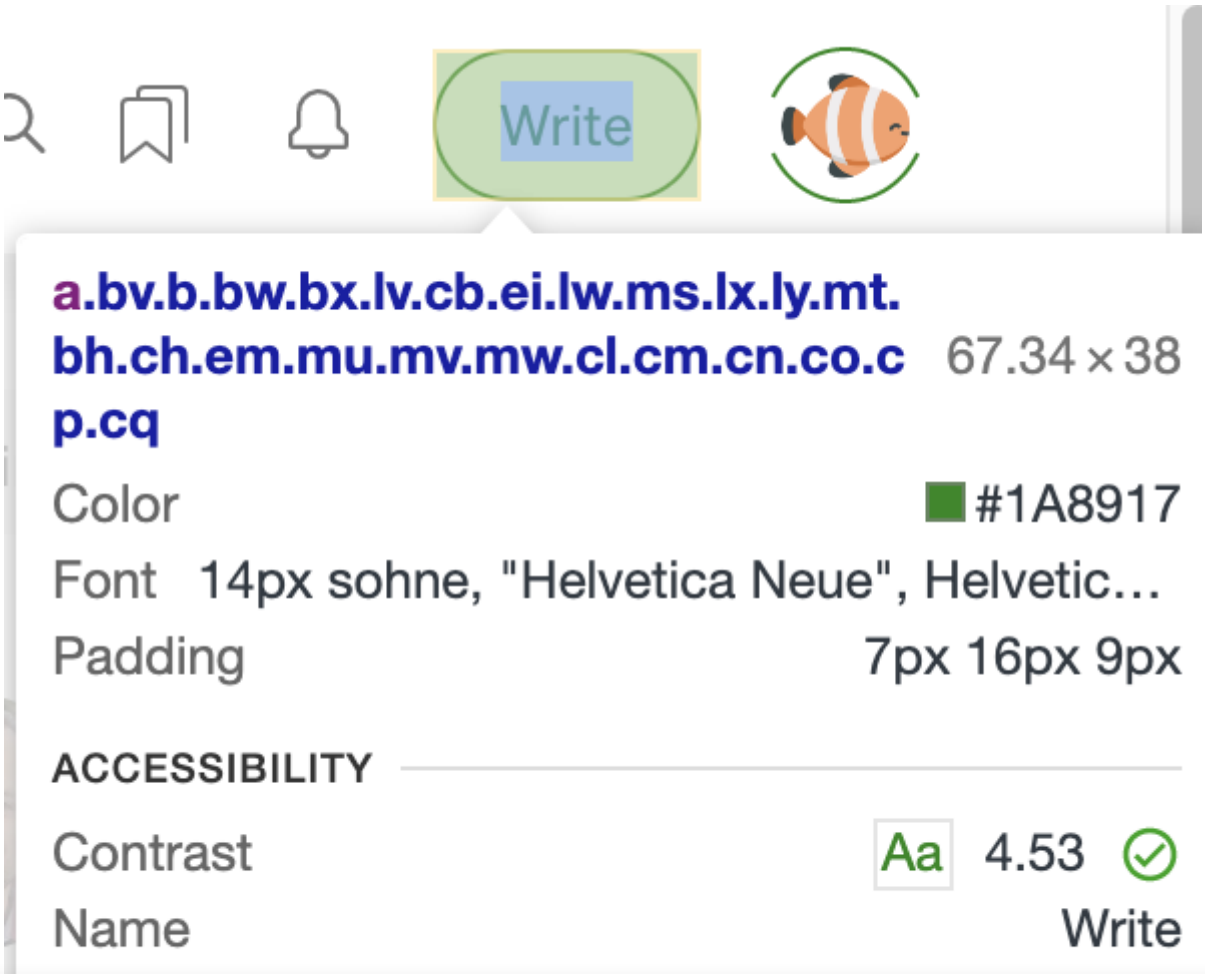
The five boxing wizards jump quickly.



We can also use the Chrome DevTool to check the color contrast of an element. Then we can find that Medium's webpage also practices this principle.



The Contrast is 14.54



Final

As the saying goes, details determine success or failure. If your project has a lot of details that can improve the user experience, you can make the user feel comfortable, and you have a higher probability of success.

Hope these tips are useful to you. Don't forget to applaud my article if you get a promotion for using these CSS tricks.